Energy-Efficient Interconnect via Router Parking [*]

Ahmad Samih[†‡], Ren Wang[ϒ], Anil Krishna[∓], Christian Maciocco[ϒ], Charlie Tai[ϒ] and Yan Solihin[‡]

[†]Intel Architecture Group
Intel Corp.
Austin, TX, USA
{ahmad.a.samih}@intel.com

[ϒ]Intel Labs
Hillsboro, OR, USA
{ren.wang, christian.maciocco, charlie.tai}@intel.com

[∓]CPU Research
Qualcomm Inc.
Raleigh, NC, USA
{krishnaa}@qti.qualcomm.com

[‡] North Carolina State University
Raleigh, NC, USA
{solihin}@ncsu.edu

Abstract

The increase in on-chip core counts in Chip MultiProcessors (CMPs) has led to the adoption of interconnects such as Mesh and Torus, which consume an increasing fraction of the chip power. Moreover, as technology and voltage continue to scale down, static power consumes a larger fraction of the total power; reducing it is increasingly important for energy proportional computing. Currently, processor designers strive to send under-utilized cores into deep sleep states in order to reduce idling power and improve overall energy efficiency. However, even in state-of-the-art CMP designs, when a core goes to sleep the router attached to it remains active in order to continue packet forwarding. In this paper, we propose Router Parking – selectively power-gating routers attached to parked cores. Router Parking ensures that network connectivity is maintained, and limits the average interconnect latency impact of packet detouring around parked routers. We present two Router Parking algorithms – an aggressive approach to park as many routers as possible, and a conservative approach that parks a limited set of routers in order to keep the impact on latency increase minimal. Further, we propose an adaptive policy to choose between the two algorithms at run-time. We evaluate our algorithms using both synthetic traffic as well as real workloads taken from SPEC CPU2006 and PARSEC 2.1 benchmark suites. Our evaluation results show that Router Parking can achieve significant savings in the total interconnect energy (average of 32%, 40% and 41% for the synthetic, SPEC CPU2006, and PARSEC 2.1 workloads, respectively).

1 Introduction

Chip Multi Processors (CMPs) are increasingly becoming the mainstream hardware architectures for a wide range of computing platforms. In such systems, the interconnect fabric consumes an increasing fraction of the chip power [29] in order to support higher bandwidths and larger number of cores. Recent studies show that, if all cores are 100% utilized, the interconnects in Intel's SCCC, Sun's Niagara [27], Intel's 80-core chip [16], and MIT's RAW chip [23] consume 10%, 17%, 28% and 36% of the total chip power, respectively. With lower core utilization, the fraction of the power consumed by the interconnect is much higher (e.g., in Intel's SCCC, interconnect power contribution jumps from 10% to 17% of the total chip power if 50% of the cores are utilized since it will continue to operate at maximum frequency to serve the active cores).

As technology and operating voltages continue to scale down, as evidenced by Intel's 22nm transistor technology [19] and the near-threshold voltage design [5], static power is becoming a larger component of the total power [2, 9, 36]. Figure 1(a) shows the breakdown of an interconnect router's dynamic versus static power at 2GHz, with V_{dd} of 1.1 V and 1.0 V, and under three different process technologies – 65nm, 45nm, and 32nm. The results are obtained from the Orion2.0 [21] power model. As shown in the figure, the static power consumption increases rapidly as technology scales down, going from 11.2% of the total router power in the 65nm (1.1V) technology to 33.6% in the 32nm (1.1V) technology.

Further, Figure 1(b) shows the breakdown of the static and dynamic power in the 32nm technology as Vdd decreases from 1.1V to 0.7V. The static power reaches 44.3% of the total power at 0.7V.

Finally, the dynamic power numbers shown in these plots are measured based on 50% load activity. According to International Data Corporation (IDC) study [15], servers achieve utilization of only 10% to 15%. With such low average utilization, the share of static power to the total interconnect power is expected to be even larger.

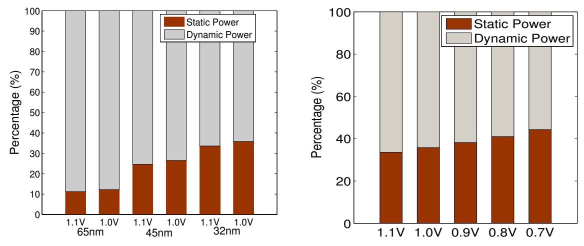


Figure 1. Static router power vs. dynamic router power for various technologies, and operating voltages. Dynamic router power is measured at 50% load.

ITRS [20] projects that by 2022, a single chip may include more than 100 cores. The rising number of cores on a single chip, accompanied by limited power envelops and low system utilization create increasing opportunities to send many of these cores to deep sleep states. Unfortunately, even after exploiting these opportunities, the system would not be energy proportional [3] if the idling power is high. With current interconnects, even when an idle core does not generate any interconnect traffic, the router and links attached to it remain active in order to maintain full interconnect operation.

[*]This work was done while the first author was a PhD candidate at NC State University. This work is supported in part by NSF grant 0915503.

Prior studies [36, 14] attempted to reduce the interconnect power by reactively power gating router components (e.g. links, ports, or buffers) in response to traffic load. With such an approach, performance may be penalized significantly if the wake-up latency is high or traffic is unpredictable. Hence, only lightweight power saving techniques can be employed. Furthermore, the approach cannot take a full advantage of situations in which a large number of cores are in deep sleep and their associated routers are idle and can be fully power gated without impacting latency by much.

To this end, and as our first contribution, we propose Router Parking (RP), a novel interconnect power management approach that aims to reduce the static interconnect power. The main idea behind our proposal is to power gate a subset of routers associated with sleeping cores by *proactively* aggregating traffic to the active routers such that it conserves power, does not impact function, and minimizes the impact on performance. Without proactive aggregation, traffic disperses across all routers even in low traffic scenarios. This can keep routers lightly loaded but active – even when their associated cores are parked. Under various topologies, such as Mesh or Torus, there usually exist multiple paths from a given source to a given destination router. Therefore, keeping all routers associated with parked cores active may not be necessary. Parking routers associated with sleeping cores must not cause interconnect partitions, where a given pair of active cores are not able to reach each other. Even after ensuring that no interconnect partitions are created, there are power and performance trade-offs in RP. The static power savings enabled by parking more routers must be balanced against the increase in interconnect latency due to rerouting around parked routers, which in turn increases run-time and energy.

In order to address this trade-off, and as a second contribution, we propose and evaluate various RP algorithms. On one end of the solution space is an Aggressive Router Parking (RP-A) approach that strives to park as many routers as possible in order to maximize static power savings. This solution, under heavy traffic, may have an undesirable end-to-end latency impact due to excessive packet detours. The other end of the solution space is a Conservative Router Parking (RP-C) approach that parks fewer, carefully selected, routers in order to minimize the impact of RP on interconnect latency. Motivated by encouraging experimental results on both ends of the spectrum, we design an Adaptive Router Parking (RP-Adp) approach that monitors the interconnect utilization and chooses between RP-A and RP-C at run time, enabling power savings while limiting performance impact.

We evaluate RP on a 64-core tiled CMP using Simics 4.7 [25] with a cycle-accurate interconnect simulator based on the Garnet [11] interconnect model and the Orion2.0 [21] power model. We experiment with both synthetic traffic and real workloads taken from the SPEC CPU2006[37] and PARSEC 2.1[4] benchmark suites. Our evaluations show that for SPEC CPU2006 and PARSEC 2.1 workloads, RP can reduce the total interconnect energy by an average of 41% and 40%, respectively. Finally, for these two workload suites, the performance metric, Weighted Speedup, is reduced by less than 0.2% and 0.4% on average, and 3% and 4% in the worst case, respectively.

2 Related Work

There is a rich body of work that has studied the problem of reducing interconnect traffic in CMPs. We divide this work into the following main categories.

Reducing Static Power Consumption. Soteriou et al. [36] proposed a component-based router power gating technique that works by monitoring the interconnect link utilization and power gating components such as ports and links in response to bursts and dips in network traffic. Although this approach shows promising result, it has several drawbacks. First, performance may suffer significantly when the wake-up latency is high or traffic is unpredictable. In order to react to changes in traffic quickly, only lightweight power saving techniques can be employed. For example, it power gates only certain router components, hence missing an opportunity to power gate entire routers. Second, the approach is agnostic to the core sleep state. It misses an opportunity for power gating mostly idle routers attached to sleeping cores. Finally, in addition to the previous issues, this approach assumes that outbound and inbound links have different physical ports for the convenience of maintaining network connectivity. Whereas many modern interconnects such as Intel's LightPeak [18] have their inbound and outbound links connected to the same physical port, and are hence either both on or both off. To the best of our knowledge, Router Parking is the first work that takes a *holistic* view of the system by *parking* selective routers based on the sleep state of their associated cores. By doing this *proactively*, traffic is aggregated away from sleeping routers, hence we avoid the need for a fast wake-up latency. At the same time, power gating entire routers enable saving more power. However, component-based power-gating may complement Router Parking (RP) if it is applied to active routers.

Reducing Dynamic Power Consumption. Matsutani et al. [26] proposed decreasing router operating frequency and supply voltage in order to reduce the router power when necessary. Link Dynamic Voltage Scaling [34] explored scaling link frequency and voltage to reduce link power consumption under low utilization. Dynamic Voltage and Frequency Scaling (DVFS) is also exploited [28] to tune the frequency of on-chip routers to improve power and performance. These approaches mainly aim at reducing the dynamic power consumed by active routers and links while our work reduces static power. This body of work is orthogonal to our Router Parking techniques and can be applied to further reduce power of the remaining active routers.

Reducing Dynamic and Static Power Consumption. Researchers have also proposed techniques to save both dynamic and static interconnect power [30, 14, 22]. Bufferless routers [30], observes that buffers consume a significant fraction of the total router power consumption, and therefore bufferless routing schemes can reduce power. These works all focus on power managing router components by reacting to the traffic pattern. In contrast, we take a holistic approach to adaptively and proactively manage the traffic and enable the entire router to enter low power state. Researchers also proposed more sophisticated topologies, e.g., Parallel Concreted Mesh (PC-Mesh) [7] and Homogeneous Parallel Concentrated Mesh (HPC-Mesh) [6] that enable richer connections thus providing more opportunities for power saving and fault tolerance.

How is Router Parking different from Fault Tolerant Networks. There has been substantial research on Fault Tolerant Networks (FTN) [1, 38, 13]. While one may imagine applying FTN techniques to route around parked routers, there are fundamental differences between FTN and our router parking. In RP, a centralized manager is preferred because it provides an ability to perform *holistic* power management. In contrast, FTN does not manage power and prefers distributed algorithms to tolerate unpredictable failures. The need for holistic power management is recognized by the industry and is incorporated into various products [17], and other NoC studies [31]. Another difference is that RP operation must be fast in order to accommodate frequent interconnect changes, while faults in FTN are rare cases. Hence, there is no need for FTN algorithms to be fast. Finally, RP must ensure full network connectivity, whereas in FTNs, faulty routers may cause disconnected network partitions.

3 Router Parking Design

At a high level, router parking is applied to a system that consists of n tiles interconnected together via an interconnect fabric with a topology such as a Mesh [1] or a Torus. Router Parking is managed by *a centralized Fabric Manager (FM)* that is responsible for configuring, monitoring and re-configuring the interconnect.

A centralized FM is essential for efficient and effective interconnect power management as seen in many recent products, e.g., Intel Sandy Bridge architecture employs a centralized System Agent to handle power management [17]. Further, changes in the interconnect configuration warrant changes in the routing tables of individual routers. This requires globally consistent view of the active network to be visible to all nodes. While this could be achieved via a distributed approach, it increases the operational complexity and reconfiguration latency as pointed out by Aisopos et al. [1].

The FM could be either added as a functionality in firmware to one of the tile's LLC cache controllers, or to one of the on-chip memory controllers. It could also be implemented as a dedicated engine located in the fabric, similar to Intel's optical fabric (LightPeak) FM [18] which holds several fabric management responsibilities, such as, setting up source-to-destination channels and handling topology changes. Further, the code of the FM could be executed on one of the cores in the system. In this work, we employ and evaluate an on-chip dedicated engine for the FM and install it in a tile positioned in the middle of the Mesh network. This ensures that the average distance to other nodes in the network is minimized.

Epochs Reconfiguration is attempted after periodic intervals called epochs. The epoch length is chosen such that $epoch\_length < core\_exit\_latency - interconnect\_reconfiguration\_latency$. For example, if the core exit latency from deep sleep state is $80\mu s$ and the latency of doing a full interconnect reconfiguration is $15\mu s$, then the epoch length should be $< 65\mu s$. This ensures that in the worst case, if a core gets interrupted immediately after reconfiguration completes, by the time it exits sleep, the interconnect has been reconfigured and its router is up and ready to service its core.

At the end of each epoch, the FM follows a sequence of actions to prepare for a potential network reconfiguration.

Gathering node information. At the end of each epoch, the FM performs a multi-cast to all *active* routers asking for the status of their associated nodes. Parked routers are guaranteed to have their associated nodes parked as well, and need not be involved in this process. Nodes with active routers reply to the FM's broadcast with information about their status (parked or active). There are three ways for parked nodes to supply their status to the FM without themselves waking up. First, a small logic in the node is responsible for communicating with the FM using the available communication links; therefore, there is no need to wake up the whole computing node to reply to the FM's query. Second, the router detects node inactivity by a timeout method and replies on behalf of the node. Third, the node notifies the FM before it goes to sleep; the FM, in turn, saves this information for future use.

Processing gathered information. Once the FM receives the node status information, it processes the information to measure connectivity and decide which routers should be parked during the next epoch. Once the FM measures connectivity, it computes routing tables for active routers using shortest path routing [2]. The FM uses the following objectives while generating a new network configuration. First, the new configuration should not partition the network. This requirement might be relaxed for special situations – for example, several functional partitions may not need to communicate with each other due to job independence. Second, the algorithm should be able to park as many routers as possible in order to achieve the highest static energy savings. Third, the algorithm should minimize the latency impact due to the necessary detours around parked routers introduced by RP. Finally, the algorithm should be fast. Section 4 describes the algorithms we evaluate in detail.

Populating new configuration. All nodes continue to follow the old network configuration and routing decisions until they receive a new configuration from the FM. Once the FM creates the bitmap corresponding to the new configuration, it sends it to all involved nodes in the system. If a given router finds that it is permitted to park, it will power gate its ports, crossbar, and arbiters. Note to ensure packet deliverability, the router may need to wait until it flushes out all packets first. If on the other hand, a given router finds that it remains active, it updates its routing table to reflect the new network configuration. Recall that the interconnect does not freeze when a transition occurs from one epoch to the next. Some routers may be operating under the old routing configuration, while others may be operating under the new routing configuration. The combination may cause undeliverable packets and deadlocks. We discuss how to handle these corner cases in Section 5.2.

4 Router Parking Algorithms

In this section, we describe two basic RP algorithms - Aggressive Router Parking and Conservative Router Parking. We then describe an Adaptive Router Parking algorithm that chooses between the two basic algorithms according to run time interconnect condition.

4.1 Aggressive Router Parking Algorithm

The goal of an Aggressive Router Parking (RP-A) algorithm is to park as many routers as possible while maintaining the network connectivity. However, it may lead to an increased packet delivery latency when packets have to travel more hops to avoid parked routers. Fortunately, there usually exists multiple minimum paths

[1] In this work we implement and evaluate our algorithms on a 2D-Mesh topology; the RP approach can, however, be applied to other interconnects with minimal modifications.

[2] Another possibility is to share the connectivity information with individual cores and let them compute their RTs.

from a given source node to a given destination. Hence, the increase in latency may not be noticeable. However, if the increase in latency is relatively large, this may increase the dynamic router energy which may offset the static energy saving from router parking. Thus, this approach is likely attractive when the interconnect packet injection rate is low because only few packets are affected and it is unlikely for them to impact performance. With a low injection rate, static energy saving may significantly exceed the additional dynamic energy consumption, making it worthwhile. Fortunately, many applications incur a low packet injection rate. For example, our analysis shows that for SPEC CPU2006 applications running on a tiled-CMP with private 32KB L1 and 1MB L2 caches, the average packet injection rate of L2 misses onto the interconnect fabric is typically low: 9 misses per 1000 instructions on average, with a maximum of 45 misses per 1000 instructions.

Under this approach, the FM views the interconnected routers as a graph; G(V,E), where routers are depicted as vertices V, connected to each other via bidirectional edges E. The FM processes the graph and evaluates its connectivity. We borrow Tarjan's Strongly Connected Components [32] algorithm from graph theory for this step. Tarjan's algorithm calculates the number of strongly connected components in a Graph(V,E) with a linear complexity of O(n); where n is the number of routers. Note that in a Mesh interconnect, *any* connected component is also strongly connected since all links are bidirectional. Therefore, we optimize Tarjan's algorithm for a Mesh topology such that we only identify the number of connected components instead of the original, heavier algorithm, that locates strongly connected components.

Algorithm 1 A pseudo-code for the RP-A algorithm

```
measure_network_connectivity()
for all $routers\ \{v\};\ that\ belong\ to\ active\ nodes\{n\}$ do
    if $v.visited$ then
        $continue$
    end if
    $strong\_cnctd\_comps(v)$
    $num\_strong\_cnctd\_comps \leftarrow num\_strong\_cnctd\_comps + 1$
    $create\ a\ new\ strong\ cnctd\ comps\ list$
    while $stack\_is\_not\_empty$ do
        $w \leftarrow pop()$
        $add\ w\ to\ new\ strong\ cnctd\ comps\ list$
    end while
end for
if $num\_of\_cnctd\_comps = 1$ then
    $return\ TRUE$    $//network\ connectivity\ is\ maintained$
end if
$return\ False$

strong_cnctd_comps(v)
$push(v)$
$v.visited \leftarrow 1$
$//Do\ a\ depth-first\ search$
for all $v's\ successors;\ \{w\}$ do
    $strong\_cnctd\_comps(w)$
end for
```

The pseudo-code of our optimized Tarjan algorithm is shown in Algorithm 1. The algorithm initially marks all routers associated with parked nodes as potential candidates for parking (except those connected to memory controllers as they provide connectivity to the main memory), and assumes they are parked. The algorithm then identifies the number of connected components. A connected component count of 1 means that parking all potentially marked routers does not partition the network. A connected component count greater than 1 means that parking all marked routers will create disjoint partitions in the network. In the latter case, the FM needs to decide how to connect the partitions. Finding the optimal path (i.e., the one that involves waking up the least number of routers) to connect the two partitions could increase the complexity of this solution to O(n^2).

In order to avoid the O(n^2) complexity, we avoid testing all potential combinations that connect partitions to find the optimal configuration. Instead, we let the FM to randomly pick an edge router (i.e. router that directly connects to a parked router) from every partition and create a path between that router and the FM itself by turning on routers along the path. Since all partitions can reach the FM, the network becomes strongly connected. Note that the chosen routers may not lead to the least amount of routers turned on. Therefore, we let the FM repeat the process k times, and then select the paths that turn on the least number of routers. How good the final solution depends on the value of k, the number of partitions, and the size of the network. In the network used in our study, we found that $k = 8$ leads to a very good configuration.

The example shown in Figure 2 helps illustrate this process. Part (a) of the figure shows a 16-tiles connected via a 2D Mesh topology. Tiles in black are parked (core is in deep sleep state). Once the FM receives the status of these cores, it marks routers associated with parked tiles as potential candidates for router parking. To apply our Tarjan-like algorithm, the FM views connected routers as a connected graph, as shown in part (b). The figure also shows that no edges connect to parked routers (black circles). After applying the algorithm, it will result in two network partitions – partition 1, 2, 3, 5, 7, 9, 13 and partition 12, 15, 16. In this case, unparking any of the routers 8, 11, and 14 can connect the partitions. Part (c) of the figure shows the two partitions connected by excluding router 11 from parking. Part (d) of the figure shows the final state of the interconnect once the new configuration is populated to all routers.

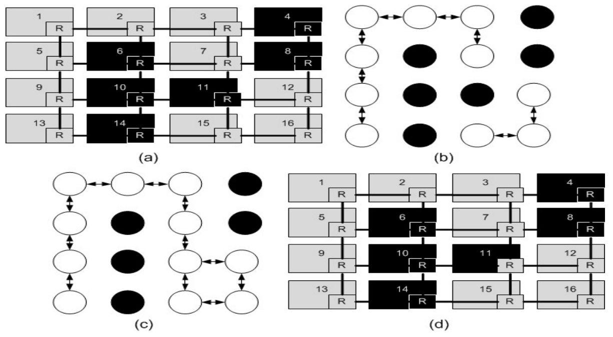


Figure 2. An example 16-node Mesh network

4.2 Conservative Router-Parking Algorithm

Under moderate to high packet injection rates, more packet rerouting due to parked routers could lead to high dynamic energy consumption which would offset the savings in the static energy achieved by RP. Conservative Router Parking (RP-C) addresses this trade-off.

RP-C relies on a subtle observation - network partitions and long detouring occur when a series of direct[3] and/or indirect[4] neighbors are parked forming a contiguous disconnect (e.g., diagonal, vertical, horizontal, staircase, loops, etc.). Such series could partition the network and introduce longer detouring latency. RP-C disallows any direct or indirect neighbor routers to be parked at the same time,

[3]East, West, North, and South neighboring routers.

[4]Northeast, Northwest, Southeast, and Southwest.

thereby avoiding both partitions and long detours. The pseudo-code for the conservative algorithm is shown in 2.

Algorithm 2 A pseudo-code for the RP-C algorithm

```
park_routers_conservatively()
for all $routers\ \{v\};\ that\ belong\ to\ parked\ nodes\{p\}$ do
    $park\_this\_router \leftarrow (any\_neighbor\_routers\_parked()\ OR\ any\_immediate\_diagonal\_routers\_parked())$
    if $park\_this\_router$ then
        $bit\_map[router\_number] \leftarrow 1$
    end if
end for
```

To understand this, let's apply RP-C to the same network as in Figure 2(a). RP-C will park a router if none of its direct or indirect neighbors is parked. If the sweep starts at tile 1, the network will end up with only 3 routers parked (4, 6, and 14); the remaining three are left on. The benefit it that the average latency between any two nodes under RP-C is much less than in RP-A. In addition to better handling high packet injection rates, RP-C is more suited to systems undergoing frequent reconfiguration. As an optimization, outer edge routers are allowed to park on series since they do not cause interconnect partitions.

4.3 Adaptive Router-Parking Algorithm

Interconnect traffic is crucial parameter in dictating whether RP-A, RP-C or no Router Parking (No-RP) is the better approach. We propose and evaluate an adaptive Router-Parking (RP-Adp) algorithm that monitors the run time interconnect utilization and chooses between these options. The algorithm is illustrated in 3.

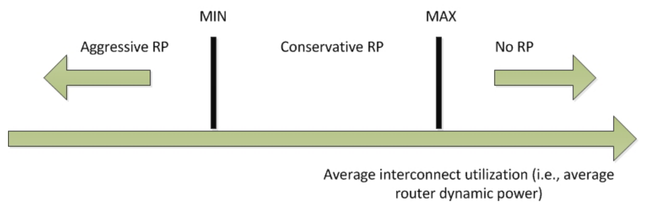

MIN
MAX
Aggressive RP
Conservative RP
No RP
Average interconnect utilization (i.e., average router dynamic power)

Figure 3. An illustration of the Adaptive Router Parking algorithm.

The average router dynamic power P_d is proportional to the interconnect utilization (number of packets switched per time unit). When P_d is smaller than a certain threshold, MIN, indicating very light traffic, RP-A is the most desirable algorithm. On the other hand, if P_d is larger than another threshold, MAX, indicating very heavy traffic (approaching the interconnect saturation point), router parking should be turned off. When P_d is between MIN and MAX thresholds, where the interconnect traffic is moderate, RP-A may not be the right choice since the reduction in static power is offset by increase in dynamic power due to heavy rerouting. Hence, the adaptive algorithm opts for the RP-C algorithm in this region.

The MAX and MIN thresholds are important parameters for the stability of RP-Adp. Here we describe how to choose the two thresholds, as well as the rationale for the choice.

Let P$_s$ denote the router's static power, which is a design parameter for a specific interconnect. *RP-A must be selected when the dynamic power introduced due to the extra hops (H_e) is less than the static power savings due to parking routers.* This can be expressed as the following inequality;

$$H_e * P_d < R_p * P_s \qquad (1)$$

where R_p is the number of additional routers parked in RP-A compared to RP-C; and H_e is the number of extra hops in RP-A compared to RP-C.

According to our design space experiments, under RP-A, H_e is usually smaller than R_p, (i.e., the number of average extra hops is usually smaller than the additional routers parked under RP-A compared to RP-C). This is because packets can usually find other minimal or near-minimal paths. Hence, if $P_d < P_s$, the inequality will be met. This means that P_s is a reasonable value for the threshold MIN. In other words, when $P_d < P_s$, RP-Adp chooses RP-A, else it chooses no-RP or RP-C (contingent on comparing P_d to MAX threshold).

When choosing the MAX threshold to switch between No-RP and RP-C, the same rationale applies. That is, if $P_d > \frac{R_p}{H_e} P_s$, router parking should be turned off. H_e corresponds to the additional hops introduced due to parking routers under RP-C compared to No-RP. R_p corresponds to the routers parked by the conservative approach.

These MIN and MAX parameters could either be chosen empirically, or at run time. In the latter case the algorithms to be compared (say, RP-C vs. No-RP) are applied in successive epochs, and the necessary statistics (average interconnect latency, average dynamic and static power, and in case of RP-C, the number of routers parked) are collected in each epoch. Applying the above inequality identifies the better algorithm for RP-Adp to employ.

5 Steady/Transient State Issues

Having looked at the architecture and algorithms associated with Router Parking at a high level, we will now focus on implementation issues that must be considered.

5.1 Deadlock Avoidance and Recovery

We use minimum routing algorithm for normal packet forwarding so that performance is not impacted significantly compared to the baseline of no router parking. However, even with low traffic rate, there exists a probability, albeit very small, for deadlocks to happen. We opt for deadlock recovery approach using an escape channel, similar to what proposed in [10]. Our approach requires minimal resources, i.e., an additional per-router flit buffer and a routing table used by the escape channel to route a deadlocked packet. Deadlock occurrence is detected by a timeout mechanism. Upon detection, a deadlocked packet is injected into a flit buffer and routed through the escape channel to its final destination. The escape channel employs up*/down* [33] deadlock-free routing algorithm. The routing tables are updated whenever an interconnect configuration takes places since the nodes comprising the escape channel may change. Details are discussed in next section.

5.2 Transient state behavior

While a reconfiguration is underway, some routers may be operating as per the old network topology, while others may be operating under the new topology. Further, if routers are allowed to park immediately once they receive a new configuration, a case may arise where some packets are rendered undeliverable. The packets cannot be routed to destination in such a case because the new topology and routing function cannot handle them. Moreover, the transition from the old to the new routing tables may create additional dependencies among network resources, causing what is referred as *reconfiguration-induced deadlock* or *ghost dependency* [24].

Many current techniques handle these problems using static configuration methods, where packet injection is prohibited and all in-flight packets have to be drained from the entire network before the new topology and configuration is deployed. Though it easily avoids any reconfiguration-induced deadlocks, it reduces network availability and system performance at each reconfiguration event. We, on the other hand, employ the following 4-phase dynamic interconnect reconfiguration protocol that does not stall the network, avoids undeliverable packets and handles the reconfiguration induced deadlock problem.

Phase 1: Update normal operation routing tables, wake up to-be-turned-on routers. (Step 1) FM wakes up routers that will be online in the next epoch. (Step 2) FM sends control packets including the new routing table to all routers that are active during the next epoch. (Step 3) Routers will update their routing tables once they receive the new reconfiguration and start operating based on the new routing information. Meanwhile, if a deadlock occurs, nodes will continue to use the *old* escape path to recover. (Step 4) FM sends a propagation order spanning tree (POST) [10] control message to all routers and awaits responses regarding the completion of the routing tables update. Once response is received, phase 2 starts. Note that, to-be-parked routers can not park after Phase 1, since they may still provide an escape path for deadlocked packets.

Phase 2: Halt injecting user-level packets onto the escape network and let it drain. Since the escape path is the only way to resolve a deadlock, it is critical to update this path without introducing deadlocks. To handle this, once Phase 1 ends, we stop using the escape path even if a deadlock occurs (i.e., postpone deadlock recovery), and drain the escape path. We initiate the drainage by issuing a Propagation Order Spanning Tree (POST) message from FM (root) to 1st order children, 2nd order children, ..., until leaves. Once the leaves receive it, they drain their up* channels and reply to their parents. Once their parents receive it, they drain their up* channels and reply to their parents and so on, until this reaches FM which then starts draining the down* channels starting the FM itself down till the leafs. This ensures that the secondary path is drained of packets.

Phase 3: Globally activate the new routing function in the escape path. The escape path has one input channel, once its drained of old packets (Phase 2) and no user-packets are being injected, the FM updates all routing tables. The update is performed at one level of the tree at a time (up*/down* tree), the FM awaits a response from the first level, then updates the second level and so on. This sequence is needed since there might be leaf nodes unreachable by their neighbors (used to be parked), so upper levels have to be updated first so these nodes are reachable. Once all levels are updated, all primary and secondary routers have been updated. Note that, phase 3 does not necessarily need to wait until Phase 2 is finished. Instead, Phase 3 can overlap with down* channels drainage (Step 2 in Phase 2).

Phase 4: Packet injection onto the escape path is allowed to resume and to-be-parked routers can park. Once all routing tables are updated, the FM signals all nodes to inform them that reconfiguration is over. The to-be-parked routers can safely park at that point, and other routers can use the escape path if they encounter a deadlock.

6 Router Parking Evaluation Methodology

6.1 System Configuration

We use a cycle-accurate multicore simulation infrastructure based on Simics [25], a full system simulation platform with a cycle accurate interconnect model based on Garnet [11], and the Orion 2.0 power model [21]. Our baseline assumes simple xy routing algorithm with no routing tables. Table 1 lists the relevant configuration parameters of the system. Throughout the evaluations, the default configuration is a 64-core (8x8) CMP connected via a 2D-Mesh.

Cores	64 in-order cores, 2GHz
Private L1 I/D, L2 caches	(L1) 32KB, 2-way, 2-cycle latency, 64B/block Private L2 caches with Capacity Sharing [8] Each private L2 cache: 1MB, 8-way, 10-cycle latency, 64B block, LRU policy
Memory	300-cycle access latency
Fabric Manager (FM)	40mW, assuming it is run on a dedicated engine. In Section 7, we show the power overhead if the FM functionality is run on one of the cores. Installed in middle tile
Routers	32nm, 2GHz @ 1.0 V$_{dd}$, 4-stage (4-cycle) pipeline, 5 I/P ports, 5 O/P ports, 4 Virtual Channels/port, 8 flits/VC, 1 place holder for deadlocked packet for escape channel, 2 routing tables, 64 entries, 3-bit/entry 10-cycle wake-up latency, overlapped with core wake-up
Links	16B, 1-cycle latency,
Energy	Router dynamic energy/access = 2.38e-10J Router static energy/cycle = 1.32e-10J static energy breakdown: Buffer = 0.9187e-10J other components (crossbar, switch arbiter, vc arbiter, and clk) = 0.402e-10J Link dynamic energy/access = 7.89103e-13J Link static energy = 0.0 Power-gating overhead = 2.3pJ
Routing algorithm	baseline: xy w/o routing tables router parking: shortest path routing for normal forwarding and up*/down* for escape channel

Table 1. System and Interconnect Configuration

6.2 Traffic Generation

We use a combination of synthetic traffic, as well as real workloads' traffic taken from PARSEC 2.1 and SPEC CPU2006 benchmark suits. Real applications tend to have low packet injection rates; synthetic traffic allows us to parameterize the injection rate and stress test RP algorithms.

6.2.1. Synthetic Traffic. For the synthetic traffic, we augment each one of the 64 nodes with a traffic generator module that injects traffic based on a statistical distribution. The destination node is selected based on the *traffic patterns* [35]. We provide a brief description of the traffic patterns in Table 2. We experiment with various fractions of parked cores (10-80%). For each run, we keep the *fraction* of parked cores constant; however, we change the *set* of parked cores every sampling period. *This allows us to introduce the reconfiguration overhead.* We run each node for 100,000 cycles. Our synthetic traffic methodology is in line with prior work [30, 29].

Traffic Distribution	Uniform (λ); λ: packet injection rate/node/cycle
Traffic Patterns	Uniform Random (UR); destination is randomly selected, Transpose (TP); (x,y) sends to (y,x) Tornado (TOR); (x,y) sends to (x+k/2 -1, y); k is dimension
Packet size	Synthetic Traffic: 2 flits/packet (e.g., 0.06 pkt/node/cycle = 0.12 flit/node/cycle) Real Workloads: 5 flits/packet (1 control + 4 data)

Table 2. Traffic Distribution Pattern

6.2.2. Real Applications. Our baseline (as shown above in Table 1) assumes private L2 caches with *capacity sharing* [8]. In this architecture, L2 victims are evicted from the local cache to a randomly selected L2 cache belonging to an unparked core, rather than being dropped off-chip. Upon a miss in the local tile, the directory is consulted and the block is supplied from the remote cache to local cache via a cache-to-cache transfer.

We construct a multi-programmed workload, consisting of 64, randomly-selected benchmarks from the C/C++/Fortran SPEC CPU2006 benchmarks [37]. Similar to the synthetic traffic, we run the workload with various fractions of parked cores, and change the *set* of parked cores every sampling period. We fast forward all active applications for 10 billion instructions. We then attach the cache models and enable detailed timing simulation. We warm each cache for 1 billion cycles, before running each application in the workload for "at-least" 150M instructions. At that point, the required statistics are collected and the application continues to run and contend for cache space.

We also experiment with 12 of the 13 benchmarks form the PARSEC 2.1 multi-threaded benchmark suite (*raytrace* is excluded due to compilation problem). In each run, we vary the number of threads of each benchmark based on the fraction of parked cores (e.g., for 10% parked cores, we run 90% (out of 64) threads rounded to the nearest integer). We create checkpoints at the start of the main work loop (PARSEC source codes clearly identifies this region). We run each thread for 150M instructions or till the end of the region of interest, whichever happens sooner.

To introduce reconfiguration overhead, the epoch length was initialized to 10,000 cycles for the synthetic traffic, and to 50,000 cycles for SPEC and PARSEC experiments. We assume short epoch length to stress test the system and to achieve many reconfigurations in a limited simulation period. This does not affect our conclusions, it only overstates the overheads of our scheme. Finally, the overhead of the Fabric Manager and the communication overhead between the Fabric Manager and the routers are included in the evaluation.

6.3 Evaluation Metrics

For both synthetic and real workloads, we compare the energy savings of the different Router Parking algorithms by measuring the interconnect static, dynamic and total energy. To measure the Router Parking performance impact on the real workloads, we measure the Weighted Speedup (WS). The significance of WS has been described in detail in prior work [12].

Finally, in order to measure the trade-off between energy savings of the interconnect (accounting for energy loss introduced by the Fabric manager and the associated communication overhead), and the performance impact on the workload, we measure the Energy-Delay Product (EDP). The delay corresponds to the total number of cycles the slowest application took to finish execution (equivalent to the number of cycles the interconnect was exercised until the slowest application finished execution). Table 3 defines these metrics.

Metric	Formula
EDP	Interconnect Energy * number of cycles
Weighted Speedup	$\sum_{i=1}^{n}(IPC_i/IPC_{i,base})$

Table 3. Performance and Energy metrics $IPC_{i,base}$ represents the IPC of an application running on core i without RP, while IPC_i represents the IPC of an application running on core i with RP.

7 Evaluation and Analysis

7.1 Synthetic Traffic Evaluation.

Significance of Static Energy. Figure 4 shows how the static energy, dynamic energy, and average network latency (y-axes) change as the packet injection rate increases (x-axis). As can be seen, the traffic injection range can be divided into four virtual regions. First, under relatively low packet injection rates (< 0.015 pkt/node/cycle, equivalent to 0.03 flit/node/cycle), the static energy consumed exceeds its dynamic energy counterpart. Second, under moderate injection rates ($> 0.015, < 0.035$), the dynamic energy starts taking over the static energy. Third, under relatively high packet injection rates ($> 0.035, < 0.060$), the dynamic energy is an order of magnitude more than static energy. Finally, under very heavy packet injection rates (> 0.060), the interconnect saturates and queuing delay starts showing up, making the average network latency exceed thousands of cycles. These observations give us insights regarding which range would be best for each RP algorithm.

For the first region, aggressive algorithm (RP-A) is more desirable since significant static energy savings can be achieved without impacting the latency and dynamic energy much; the amount of traffic being detoured is small. At the other extreme, when the interconnect is saturated, any RP algorithm would make the situation worse; hence, RP should not be applied and all routers should be active.

The regions in the middle requires trade-off analysis. For example, for relatively high packet injection rates – say 0.06 pkt/node/cycle (or alternatively 0.12 flit/node/cycle), RP-A may not be the best option since static energy savings by parked routers can be easily offset by an increase in dynamic energy due to heavy traffic rerouting. Hence, conservative algorithm (RP-C) could potentially be more appealing in this case. On the other hand, if packet injection is, say, 0.02, then the increase in dynamic energy due to rerouted traffic could still be less than the static energy savings, giving the RP-A an appealing use case. RP-Adp can decide the winning algorithm. We evaluated a wide range of packet injection ratios in each of the four regions and found that the results in each region are consistent. Due to space limitations, throughout the upcoming sections, we restrict the results to three packet injection scenarios – 0.01 (static energy $>$ dynamic energy), 0.04 (static energy $<$ dynamic energy), and 0.06 (static energy $<<$ dynamic energy).

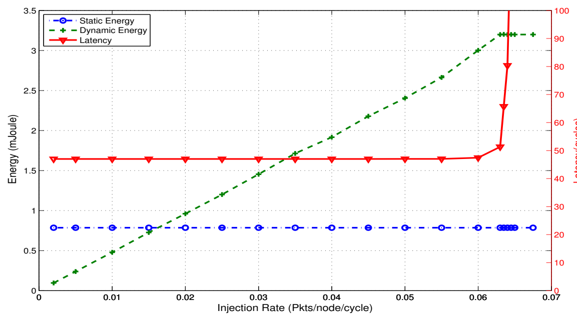


Figure 4. Interconnect static energy, dynamic energy, and average latency for uniform random traffic pattern, for various packet injection rates; each packet is 2 flits.

In-depth Analysis of Router Parking Algorithms. Figure 5(a) shows the distribution of 40% randomly parked cores (in black) in the 8x8 2D Mesh network. As shown, letting all parked cores park their routers may cause network partitions, hence the need for intelligent Router-Parking algorithms. Parts (b,c,d) show the heat maps for the 8x8 router utilization under the No-RP, RP-A, and RP-C, for the same 40% parked cores ratio, and 0.04 packet injection rates. The color of a tile corresponds to the average router utilization, with bright colors indicating higher router utilization, dark colors indicating low router utilization, and black colors indicating no traffic going through. As shown in the No-RP, since all routers are active, the average utilization is low, as depicted by relatively dark-colored routers. Note that, even in the No-RP case, with the 40% fraction of parked cores, some middle routers are darker since they are not generating traffic but merely forwarding packets for other nodes. When RP-A is applied (part c), black routers indicate that they are parked. Further, since the number of active routers is less than No-RP, the average utilization is higher as shown by the bright colors. Moreover, since in RP-A, a cluster of routers can be parked as long as no network partitions are created, long detours may be introduced. This can be visualized around the cluster of routers in middle of network. Under the RP-C case (part d), the number of parked routers is not as many, hence, the average utilization is in between no-RP, and RP-A cases. Further, the RP-C figure shows that no clusters of parked routers are created. This leads to little latency impact in RP-C case as compared to RP-A.

Figure 6 shows three rows for different packet injection rates (top row 0.01, middle row 0.04, and bottom row 0.06 pkt/node/cycle). Each row shows the interconnect static energy, dynamic energy (including the FM overhead), average latency, and the total (static + dynamic) interconnect energy normalized to the base case of no Router Parking. The x-axes correspond to the fraction of parked cores. The figures compare the three algorithm RP-A, RP-C, and RP-Adp to the base case of no Router Parking (no-RP).

Several interesting observations can be drawn from these figures. First, RP-A - due to its ability to park many routers, achieves significant static energy reduction for the three packet injection scenarios. Further, this reduction is accompanied by modest dynamic energy increase for the low packet injection rate case (top row), hence, resulting in significant *total* energy savings (RP-A reduces total interconnect energy by an average of 32% and up to 61% compared to no-RP case). However, as the packet injection rate increases (from middle to bottom row), *and* the fraction of parked cores is below 30% for 0.04 pkt/node/cycle and below 40% for 0.06 pkt/node/cycle, the increase in the dynamic energy partially offsets the reduction in the static energy, hence, increasing the total interconnect power slightly (as shown in part d). As the fraction of parked cores grows, the static energy reduction becomes more visible and leads to total energy savings. For example, even for high packet injection rates (bottom row), RP-A still reduces *total* interconnect energy by an average of 8%.

Second, as mentioned in Section 3, RP-A parks more routers than RP-C, hence, it has a bigger impact on interconnect latency (part c) due to longer detours. This produces heavier traffic rerouting, which may lead to an increase in the dynamic energy offsetting any savings in the static energy. This indicates that RP-A is less attractive when the traffic injection rate is high and the fraction of parked cores is small. In such cases, RP-C is more favorable.

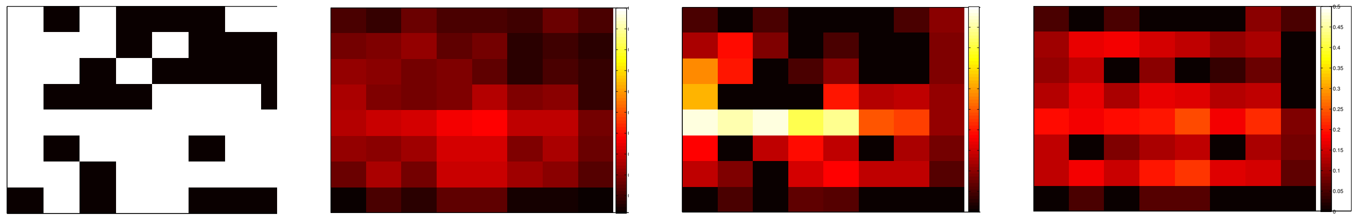
Therefore, neither RP-A nor RP-C can fit the wide spectrum of run-time dynamics. This is why we need an adaptive design that can choose either one dynamically, based on the run-time network conditions. For example, with low injection rate (top row), and since the overall network traffic is light and dynamic energy is small, RP-Adp selects RP-A in order to park as many routers as possible for power saving (RP-Adp is overlaid on top of RP-A in the top row of the figure). On the other hand, with packet injection rate of 0.06 (bottom row), when the parked nodes are few (ratio below 10%), with more active nodes injecting traffic, average traffic in the network is relatively heavy. In this case, RP-Adp closely follows no-RP behavior. As the ratio of parked nodes gradually increases, indicating average traffic in the network becomes progressively lighter, we can see that RP-Adp first diverts (around 10% to 20% ratio) from no-RP to RP-C. Then as the ratio increases further(from 60% to 80%), indicating even fewer injecting nodes and lighter traffic in the network, RP-Adp gradually converges to RP-A. These set of results show that our RP-Adp can capture the network dynamics and reacts accordingly, achieving the design objectives with significant gains (an average of 32%, 19%, and 11%, for the three packet injection rates, respectively) and almost no reduction in total interconnect energy in the case of high number of active nodes accompanied by high traffic loads, as shown in Figure 6(d).

Finally, although the average interconnect latency impact may seem high with large fraction of parked cores, (32% increase in latency for RP-A under 60% fraction of parked cores as shown in part c), it does not necessarily transform in performance slowdown. The reason is that it highly depends on the amount of traffic being rerouted. For example, if a given workload posts very few interconnect requests over its entire run time, even if these requests take a long detour, this will not impact the application's performance by much. Hence, the lower the packet injection rate, the lower the performance impact. We will show in the next section, that with typical workloads (SPEC CPU2006 and PARSEC2.1), the observed impact on performance is limited due to the fact that these workloads have relatively low injection rates and the interconnect is not the bottleneck.

7.2 Real Workload Evaluation

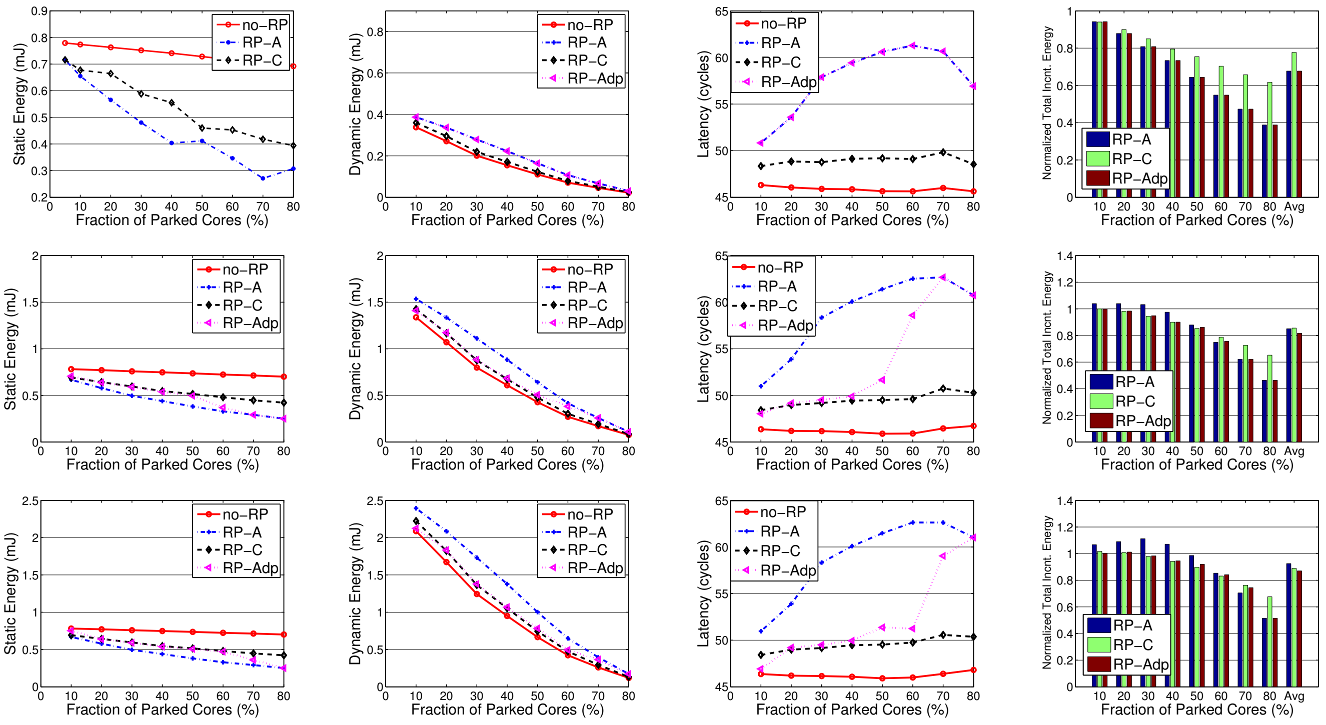
Figure 7 shows the various metrics for the SPEC CPU2006 workload mix, running RP-A, RP-C, and RP-Adp, normalized to the base case of no Router Parking. The x-axes in the figures show the varying fraction of parked cores. As shown in part (a) of the figure, RP-A and RP-C achieve significant *total* interconnect energy reductions reaching up to 61% and 42% for 80% fraction of parked cores, respectively. On average, RP-A and RP-C can achieve 40%, and 25% reduction in total interconnect energy. The reason why RP-A consistently outperforms RP-C in reducing total interconnect energy is due to the marginal increase in interconnect dynamic energy RP-A introduces while parking more routers. Part (b) of the figure confirms this observation by showing the Weighted Speedup normalized to the base case of no-RP. As shown, the performance is impacted by less than 0.2%. Hence, the associated dynamic energy increase is expected to be low. The normalized EDP (part c) follows a similar trend to the normalized energy reduction since the delay is marginally impacted showing significant reduction in EDP. RP-Adp was able, at run time, to detect this and pick RP-A.

Recall that weighted speedup is calculated across all 64 applications in the workload. To gain better understanding into how RP impacts individual applications, the figure part (d) shows a memory intensive applications out of the 64 applications comprising the



(a) Core on/off (b) No-RP (c) RP-A (d) RP-C

Figure 5. Core on/off bitmap (a), Heat Maps for the 2D Mesh, 64 routers (Brighter is higher temperature), under the three cases, No RP (b), RP-A (c), and RP-C (d). Heat in the figure corresponds to router utilization. Traffic assumed is uniform random.



(a) (b) (c) (d)

Figure 6. Interconnect static energy (a), interconnect dynamic energy (b), average interconnect latency (c), and total (static + dynamic) interconnect energy normalized to the base case of no Router Parking (d) for packet injection rates of 0.01 (top row), 0.04 (middle row), and 0.06 (bottom row) pkt/node/cycle (or alternatively 0.02, 0.08, and 0.12 flit/node/cycle). Traffic assumed is uniform random.

.

workload mix (i.e, *mcf*). *mcf* is impacted the most across all applications. This is expected since *mcf* posts many interconnect misses (MPKI=45) and therefore the interconnect latency might become a bottleneck. *mcf* notices slowdown of up to 3%. This is compared to another non-memory intensive application; i.e., part (e), *namd* (MPKI=0.93), which shows almost no noticeable impact since it does not post as many interconnect misses. Many SPEC CPU2006 applications follow *namd* behavior. *To summarize, interconnect latency is not the bottleneck for many SPEC CPU2006 workloads. Hence, although RP increases average interconnect latency, it does not impact the applications' performance significantly.*

Similar trends are shown for the PARSEC 2.1 workload as depicted in Figure 8. On average, RP-Adp achieves 41% reduction in total interconnect energy with almost less than 0.4% slowdown in Weighted Speedup. Hence achieving an average reduction of 38% in EDP. *canneal* (part d) was the application impacted the most by Router Parking. It is slowed down by up to 4%. Similar to above, RP-Adp picked the RP-A as a run-time choice since the savings in static energy offset the dynamic energy increase.

Impact of prolonged execution on energy savings. It is critically important that the slowdown in application's performance (and the corresponding increase in processor run time) does not have an offsetting impact on energy. In order to illustrate this, Figure 9 (a,b) shows the total interconnect energy for RP-Adp with and without considering the extra energy consumed by active cores, normalized to the base case of no Router Parking, for SPEC CPU2006 and PARSEC2.1 workloads, respectively. The cores considered in this experiment are similar to those existing in Intel's SCCC with 2W TDP/core. As shown in the figure, the energy impact due to cores running longer time had negligible impact on the overall intercon-

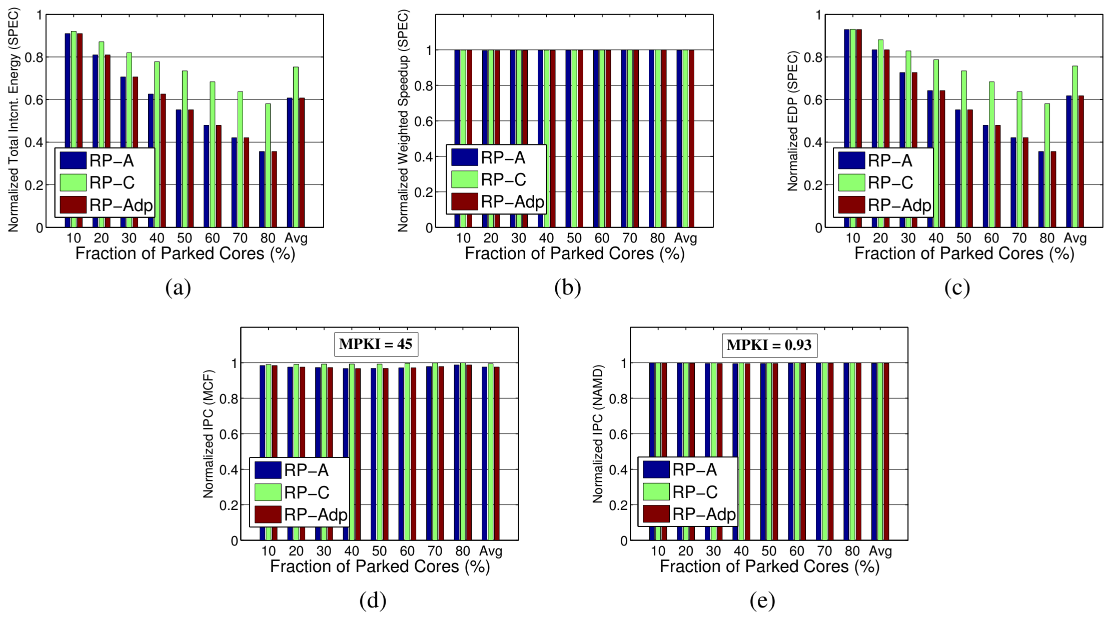


Figure 7. Total interconnect energy (a), Weighted Speedup (b) and EDP (c) for the SPEC CPU2006 workload. IPC Speedup for one memory intensive application *mcf* (d), and IPC Speedup for a non-memory intensive application *namd* (e). Each figure shows RP-A, RP-C and RP-Adp with varying fraction of parked cores, with the average being the last set of bars in each figure. All are normalized to the base case of no Router Parking.

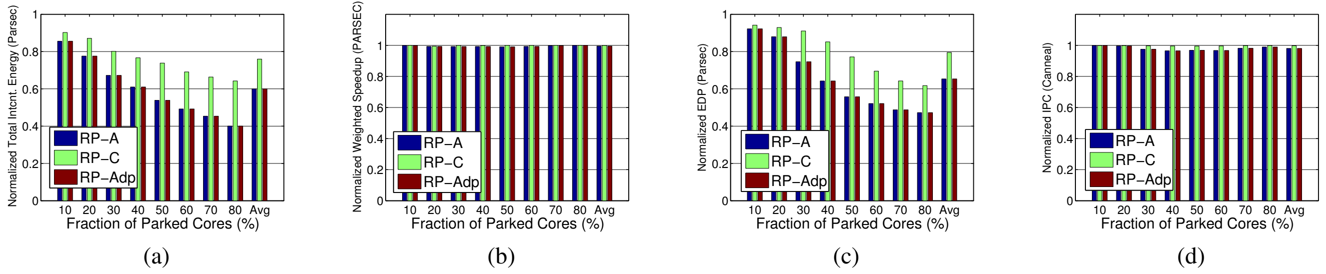


Figure 8. Total interconnect energy (a), Weighted Speedup (b) and EDP (c) for the PARSEC2.1 workload. IPC Speedup for a memory intensive application *canneal* (d). All are normalized to the base case of no Router Parking.

nect energy savings. This shows the potential of RP to reduce *total chip* energy.

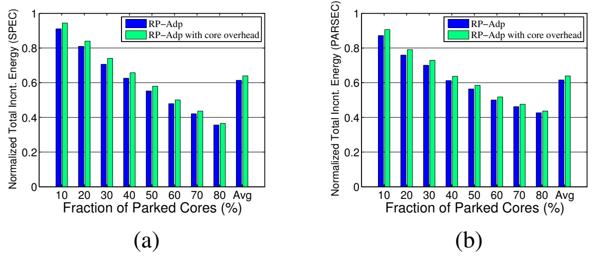
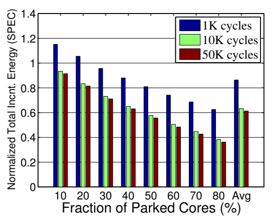


Figure 9. Total interconnect energy for RP-Adp, with and without extra core energy consumed by longer runs, for SPEC CPU2006 (a), and PARSEC2.1 (b), normalized to the base case of no Router Parking.

Impact of epoch length on energy savings. Figure 10 illustrates the interconnect energy of RP-Adp for SPEC CPU2006 and PARSEC 2.1 workloads, normalized to the base case of no Router Parking, under different epoch lengths; 1k cycles, 10k cycles and 50k cycles. Recall that the epoch length depends on the core exit latency and the reconfiguration latency(Section 3). As shown in the figures, for very short epoch length (1000 cycles) - albeit unrealistic - the energy consumed while re-configuring the interconnect is significant such that, for small fraction of parked cores, it not only offsets the interconnect energy savings, but also increases the interconnect energy compared to the base case of no Router Parking by up to 18%, making router parking undesirable in environments with such reconfiguration frequency. As the epoch length increases, the energy overhead becomes relatively small, hence, it is easily offset by energy savings.

7.3 Sensitivity Analysis

Sensitivity of Router-Parking Algorithms to Various Traffic Patterns. In Section 7.1, we analyzed the Router Parking algorithms behavior under uniform random traffic pattern. In this section, we evaluate Router Parking algorithms (RP-A, RP-C and RP-Adp) under different traffic patterns.


(a)

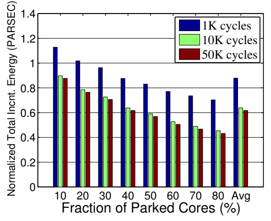

(b)

Figure 10. Total interconnect energy for RP-Adp with various epoch lengths, for SPEC CPU2006 (a) and PARSEC 2.1 (b), normalized to the base case of no Router Parking.

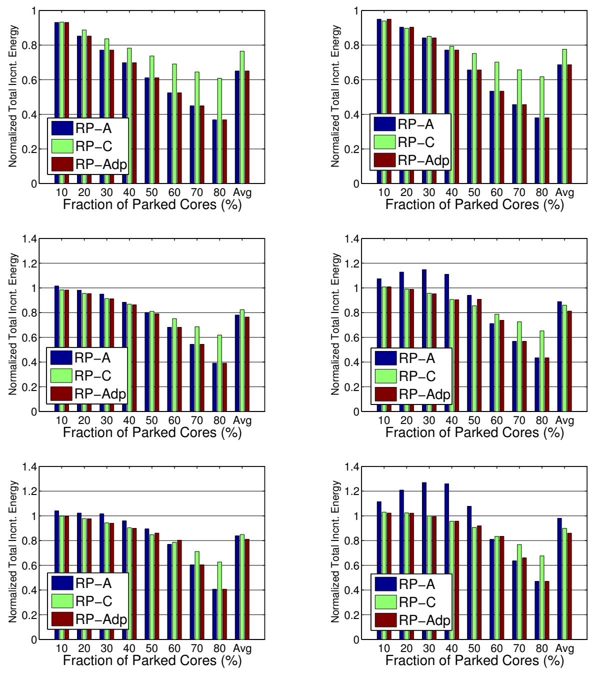

(a) TP (b) TOR

Figure 11. Total interconnect energy for different injection rates 0.01 (top row), 0.04 (middle row), and 0.06 (bottom row) for Transpose (TP) and Tornado (TOR) traffic patterns, normalized to the base case of no Router Parking.

Figures 11 (a)(b) show the total interconnect energy under two different traffic patterns (TP and TOR), for various packet injection rates (top row 0.01, middle row 0.04. bottom row 0.06), normalized to the base case of no Router Parking. The major takeaways from this series of data are the following:

First, the traffic pattern plays an important role in how well each Router Parking algorithm performs. For Tornado (TOR) traffic pattern, each packet has only one single minimal path. Any parked router between the source and destination causes packets detour. The situation is even more severe with RP-A, which may configure adjacent routers to be parked, making a packet travel several extra hops. On the other hand, For Transpose (TP), there are multiple minimal paths between source/destination pairs, (e.g, 9 minimal paths from (1,3) to (3,1)). Which means, even if RP algorithms park some routers in between, there is still a good chance that packets will follow another minimal path so the latency is not increased. Due to this reason, with the same injection rate, both RP-A and RP-C perform better under TP traffic pattern than TOR pattern, especially RP-A.

Second, the results show that under high packet injection rates (e.g., 0.06) with small fraction of parked cores ($< 30\%$), if the default latency is small (e.g., TOR-like traffic pattern), the default policy of no-RP is more efficient than applying either RP-A, or RP-C (Figure (c) bottom row). Thus, our adaptive algorithm (RP-Adp) adapted to such condition and produced negligible impact on interconnect energy.

Sensitivity to running on a dedicated engine vs. on one of the cores. As discussed in Section 3, the FM functionality could be run on a dedicated engine (our baseline assumption), or it could be run on one of the available cores. These two approaches have several advantages and disadvantages. While running on a dedicated engine is much faster and consumes less power, it requires changes to the hardware and introduces area overhead. On the other hand, running on one of the cores does not require major hardware changes, however, at the cost of slower operation and higher power consumption. Our evaluation shows that running on a dedicated engine has almost negligible power overhead if compared to an ideal FM. Further, running on one of the cores (2W TDP/core) reduces the average interconnect savings from 40% to 37% on average. This shows that, from power overhead perspective, both approaches are fruitful.

8 Other Issues

Shared LLC CMPs. Many contemporary CMPs are designed with a *distributed but shared* Last Level Cache (LLC). For routers in such CMPs, even if a core is parked, its slice of the distributed LLC and the associated router cannot be power gated. This is because the contents of the cache slice must be retained to service mapped cache requests and maintain cache coherency. Although Router Parking does not apply to such architectures *directly*, Router Parking can be augmented with a distributed LLC power management policy. This power management policy power-gates selective LLC slices dynamically by changing the hash function that determines the home LLC slice. This is outside the scope of this paper and we leave it as future work.

Scalability The Fabric Manager does not necessarily need to communicate with all nodes in the system following a network reconfiguration. It could perform localized network re-configuration such that it limits the reconfiguration overhead. Further, online connectivity algorithms such as Online Tarjan algorithm can re-evaluate the connectivity of the interconnect if a few nodes change their status without having to reevaluate the entire network from scratch. Employing localized reconfiguration and online algorithms improve the scalability of the centralized Fabric Manager. Finally, with a much larger number of cores, an *n*-level hierarchy of Fabric Managers could be introduced, with each FM taking responsibility of an island of cores. All first level FMs connect to a second level FM, and so on. Recall that as the number of cores grows, the opportunity of core parking grows, hence larger static energy savings are realized. We believe that RP will continue to be a fruitful architecture for future many-core CMPs.

What if there are no routers available to park? In case there are no available routers to park (i.e., all cores are active), RP architecture would consume more power than the baseline case due to the additional routing tables introduced in each router. Such routing tables are not needed in a 2D-Mesh interconnect that uses dimension-ordered routing (e.g., XY). In such a case, the FM can send out notifications to all nodes to turn off their routing tables, and revert to XY routing.

9 Conclusions

We proposed and motivated Router Parking (RP), which selectively power gates interconnect routers in large Chip Multi Processors (CMPs), as a step towards energy proportional computing. We evaluated three RP algorithms - an aggressive approach (RP-A) that parks as many routers as it can while maintaining connectivity between active cores, a conservative approach (RP-C) that tries to reduce the performance impact of rerouting traffic around parked routers, and an adaptive approach (RP-Adp) that dynamically selects the better of the first two approaches. RP-A works better for light traffic because it gets the most static power savings with only a small performance impact due to increased packet routing latency. RP-C parks fewer routers in order to keep detours around parked routers short. RP-C works better as the traffic increases - it ensures that the dynamic energy consumed due to the longer detour latency does not overtake the static energy savings due to parked routers. We experimented with both, synthetic traffic of different patterns, and real workloads taken from the SPEC CPU2006 and PARSEC 2.1 benchmark suits. Overall, we found that RP reduces the total interconnect energy by 32%, 40% and 41% respectively, while only reducing the performance by less than 0.5%.

10 *

References

[1] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco. Ariadne: Agnostic reconfiguration in a disconnected network environment. PACT '11, Washington, DC, USA, 2011.

[2] A. Banerjee, R. Mullins, and S. Moore. A power and energy exploration of network-on-chip architectures. NOCS '07, pages 163–172, Washington, DC, USA, 2007.

[3] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, 2007.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. Tech. Rep. TR-811-08, Princeton University, 2008.

[5] S. Borkar and A. A. Chien. The future of microprocessors. *Commun. ACM*, 54:67–77, 2011.

[6] J. Camacho and J. Flich. Hpc-mesh: A homogeneous parallel concentrated mesh for fault-tolerance and energy savings. ANCS'11, Washington, DC, USA, 2011.

[7] J. Camacho, J. Flich, A. Roca, and J. Duato. Pc-mesh: A dynamic parallel concentrated mesh. In *Parallel Processing (ICPP), 2011 Intl. Conf. on*, pages 642 –651, 2011.

[8] J. Chang and G. S. Sohi. Cooperative Caching for Chip Multiprocessors. In *ISCA'06*, 2006.

[9] X. Chen and L.-S. Peh. Leakage power modeling and optimization in interconnection networks. ISLPED '03, pages 90–95, New York, NY, USA, 2003. ACM.

[10] J. Duato and T. Pinkston. A general theory for deadlock-free adaptive routing using a mixed set of resources. *Parallel and Distributed Systems, IEEE Trans. on*, 12(12):1219 –1235, 2001.

[11] N. A. et. al. GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator. 2009.

[12] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *IEEE Micro*, 28(3), 2008.

[13] M. Gomez, J. Duato, J. Flich, P. Lopez, A. Robles, N. Nordbotten, O. Lysne, and T. Skeie. An efficient fault-tolerant routing methodology for meshes and tori. *Computer Architecture Letters*, 3(1):3, 2004.

[14] H. Matsutani, et al. Performance, area, and power evaluations of ultrafine-grained run-time power-gating routers for cmps. *CAD of Integrated Circuits and Systems, IEEE Trans. on*, 30(4):520 –533, 2011.

[15] IDC. Future of Virtualization. *http://www.vmware.com/files/pdf/analysts/Future-of-virtualization-IDC.pdf*, 2008.

[16] Intel Corp. Teraflops Research Chip. *download.intel.com*, 2007.

[17] Intel Corp. Core Micorarchitecture. *http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.9-Desktop-CPUs/HC23.19.911-Sandy-Bridge-Lempel-Intel-Rev%207.pdf*, 2008.

[18] Intel Corp. Thunderbolt Technology. *http://techresearch.intel.com/spaw2/uploads/files/*, 2011.

[19] Intel Pressroom. Intel 22nm 3-d tri-gate transistor technolog, 2011.

[20] ITRS. International Technology Roadmap for Semiconductors: 2008 Edition, Assembly and packaging. In *http://www.itrs.net/Links/2008ITRS/AP2008.pdf*, 2008.

[21] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. 2009.

[22] G. Kim, J. Kim, and S. Yoo. Flexibuffer: Reducing leakage power in on-chip network routers. In *Design Automation Conf. (DAC), 2011 48th ACM/EDAC/IEEE*, pages 936 –941, 2011.

[23] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff. Energy characterization of a tiled architecture processor with on-chip networks. ISLPED '03, New York, NY, USA, 2003. ACM.

[24] O. Lysne, J. Montanana, J. Flich, J. Duato, T. Pinkston, and T. Skeie. An efficient and deadlock-free network reconfiguration protocol. *Computers, IEEE Trans. on*, 57(6):762 –779, 2008.

[25] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *Computer*, 35(2), 2002.

[26] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano. Adding slow-silent virtual channels for low-power on-chip networks. In *NoCS 2008. 2nd ACM/IEEE Intl. Symp. on*, pages 23 –32, 2008.

[27] H. McGhan. Niagara 2. *Microprocessor Report*, 2006.

[28] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. MICRO 42, New York, NY, USA, 2009. ACM.

[29] A. K. Mishra, N. Vijaykrishnan, and C. R. Das. A case for heterogeneous on-chip interconnects for cmps. ISCA '11, New York, NY, USA, 2011. ACM.

[30] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. ISCA '09, New York, NY, USA, 2009. ACM.

[31] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip networks from a networking perspective: congestion and scalability in many-core interconnects. In *ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 407–418, New York, NY, USA, 2012. ACM.

[32] R. Tarjan. Algorithm Design. *Communications of the ACM*, 1987.

[33] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *Selected Areas in Communications, IEEE Journal on*, 9(8):1318 – 1335, 1991.

[34] L. Shang, L.-S. Peh, and N. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *HPCA-9 2003.*, pages 91 – 102, 2003.

[35] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Goal: A load-balanced adaptive routing algorithm for torus networks. In *Intl. Symp. on Computer Architecture (ISCA*, pages 194–205, 2003.

[36] V. Soteriou and L.-S. Peh. Exploring the design space of self-regulating power-aware on/off interconnection networks. *IEEE Trans. Parallel Distrib. Syst.*, 18:393–408, 2007.

[37] Standard Performance Evaluation Corporation. *http://www.specbench.org*, 2006.

[38] J. Wu. A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model. *Computers, IEEE Trans. on*, 52(9):1154 – 1169, 2003.